Inline Cache for Memory-Intensive Algorithm Acceleration via HLS

Liang Ma
Supervisor: Prof. Luciano Lavagno

Research context and motivation

- High-performance computing (HPC) via the accelerators e.g. GPUs, FPGAs.
- Balancing the execution speed, energy consumption and developing time and efforts to accelerate the algorithms properly on an FPGA when compared to other accelerators.
- Target to accelerate the memory-intensive algorithms which exchange large data arrays with external DRAM. The bandwidth from the kernel to the external DRAM is usually the bottleneck of the FPGA as an accelerator.
- Traditionally, there are two aspects could be considered in order to accelerate the kernel.
  1. One is to implement a wider AXI interface (via user-defined datatype, vector type, structure, etc.) in order to read or write multiple bytes of data in a single clock cycle.
  2. The other one is to exploit the on-chip Block-RAMs to store the data frequently used by the kernel. The BRAMs are able to be re-sized or partitioned according to the access pattern in order to achieve parallel accesses to the memory.
- However, for each of the two optimizations, the programmers have to build the control units to manage the accesses to the on-chip memory or to manage the read and write at the function interfaces. It requires the programmers to analyze the details of the algorithm and the data structure and to make lot of changes to original codes.

Addressed research questions/problems

- Concerning the problems above, we wonder whether it is possible to find a general-purpose solution that will approximate the speed of a dedicated manually written data movement solution to local memory.
- The cache mechanisms is implemented at C++ level in order to find a solution that
  1. Significant improve the performance and the power efficiency
  2. Enable the use of HLS tools
  3. Keep the standard HLS-based verification flow
  4. Require almost no changes to the original algorithms
  5. Not hampering the standard set of optimizations, architectural choices
- This solution has been tested in many applications from various fields such as: matrix multiplication, KNN, Bitonic sorting, Smith-waterman method and Lukas-Kanade method

Future work

- The future work will focus on automating application profiling in order to select the best cache architecture for each DRAM array static address analysis in order to infer which accesses are always "hits"
- The automation could be based on the deep learning technologies

Novel contributions

- Separate cache for each array to minimize the conflicts and to improve the efficiency
- Caches implement C++ ’[] ’ operators, in order to require minimal source code changes
- Various types of caches in order to best optimize their architecture
- Automatically adapt each global memory array mapped to the caches using wide memory interfaces in order to optimize transfer bandwidth to the external DRAM
- Enable verification of cache performance and correctness using the standard C++-based verification flow supported by modern HLS tools
- Enable the loop optimizations which can be made by the HLS tools only for the arrays mapped to on-chip BRAM, and not to off-chip DRAM

Adopted methodologies

- The caches are modeled as templated C++ classes supported by modern HLS tools.
  1. Direct-mapped cache
  2. Set-associative cache
- Other types: read-only, write-only cache, etc.
- Design flow and architecture:

Optimization and results (of the Lukas - Kanade algorithm):

The energy reported here doesn't include the part of the external memory

Submitted and published works

- L. Ma, L. Lavagno, M. T. Lazarevski, A. Afr., Acceleration by Inline Cache for Memory-Intensive Algorithms on FPGA via High-Level Synthesis,” in IEEE Access, 2017 (Accepted, available online)

List of attended classes

- 02LHRV – Communication (16/02/2017, 1)
- 01RLRV – Complex Networks: Models and Applications (21/06/2016, 4)
- 01OTEIU – Data mining concepts and algorithms (11/10/2016, 4)
- 01GPOIU – Exponential Algorithms for Combinatorial Optimization (10/02/2016, 4)
- 01NDLRV – Lingua italiana livello (20/01/2016, 3)
- 01WNWQO – Modeling and optimization of embedded systems (29/01/2016, 6)
- 02JEVLJ – Object oriented programming (27/01/2017, 6)
- 08IXTRV – Project management (20/09/2016, 1)
- 01RISRV – Public speaking (16/02/2017, 1)
- 01GSCIU – Reconfigurable computing (10/03/2016, 4)
- 01RMRV – Short Course on Entrepreneurship (23/11/2015, 1)
- 01QURV – The redefinition of the International System of Units (11/03/2016, 3)
- 01QORRV – Writing Scientific Papers in English (10/06/2016, 3)